MoisturEC version 1.0

MoisturEC is an R-based GUI used to combine electrical conductivity (EC) data with point moisture measurements for an updated moisture estimate capitalizing on the accuracy of point moisture measurements and the spatial coverage of EC data. It is assumed that the EC data have been inverted already and that EC values correspond to points in space.

Point moisture information is usually accurate but spatially sparse. EC data from geophysics tends to have good spatial coverage by comparison, but lacks direct sensitivity to changes in moisture content, particularly at increasing distance from the geophysical measurement point.

This help file goes through the practical steps of using moisturEC, as well as details of the computations used. Further help is available by hovering the mouse over inputs within the GUI.

At a minimum, two data files are required: one with EC data, and one with point moisture data. The EC data is converted to moisture, either using an Archie relation (default), using collocated point moisture/EC data values, or by user uploaded calibration data. The user can then click "calculate moisture" to combine the estimates through an optimization of a tradeoff parameter. The results are displayed in a second tab in the GUI. Results are also output as text and .vtk files (for visualization in Paraview).

System requirements

MoisturEC has been tested on a Windows system with R version 3.4.3. Linux and Mac operating systems are untested, as are earlier versions of R.

Installation

[1] Download and install R (https://www.r-project.org/) and (recommended) Rstudio (https://www.rstudio.com/)

[2] open R and install the needed packages. This can be done with the following commands:

```
#downloads and installs needed packages

list.of.packages = c('shiny','shinyBS','shinyjs','ggplot2','viridis',
'plotly','Matrix','pracma')

new.packages = list.of.packages[!(list.of.packages %in%
installed.packages()[,"Package"])]

if(length(new.packages)) install.packages(new.packages,dependencies = T, repos
= "https://mirrors.nics.utk.edu/cran/")
```

[3] Unzip the mEC_Lite folder, open the "mEC.r" script with Rstudio, and run the app (current working directory must be mEC_Lite folder).

Overview

What does moisturEC do?

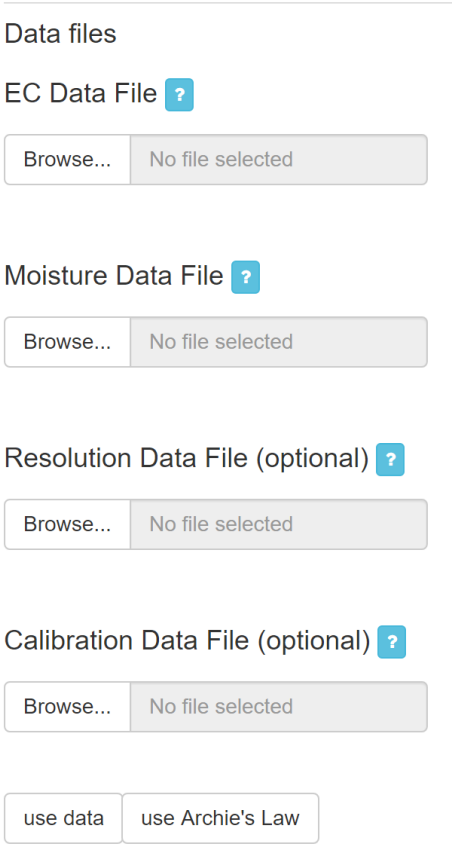
- Combines 2D or 3D electrical conductivity (EC) and point moisture data into single moisture estimate
- Is open source allowing the user to view or modify the code
- Provides convenient, visual framework to estimate moisture from EC data, with three options for EC-moisture conversion
- Provides numerous interactive plots of data and results
- Determines appropriate level of smoothing of the final estimate based on user supplied information on data error, parameter error, and resolution.
- Outputs plots and text files of results

What does moisturEC NOT do?

- Invert raw geophysical data
- Interpolate/krige point moisture information alone

Inputs

Data files:

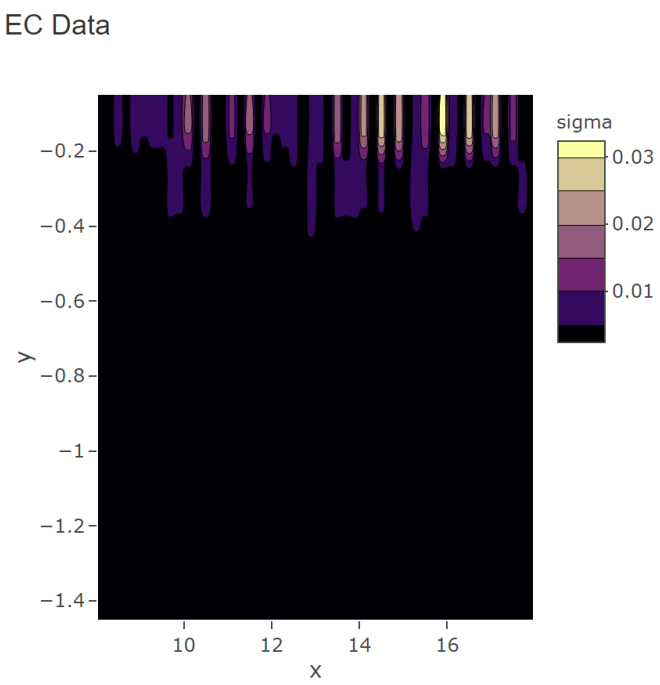


Electrical conductivity (EC) data and point moisture data are uploaded via the GUI. Optional input of a resolution data file and a calibration data file are also available.

EC Data file

A .csv file containing a header with columns X, Y, Z, EC (S/m) and % error in EC

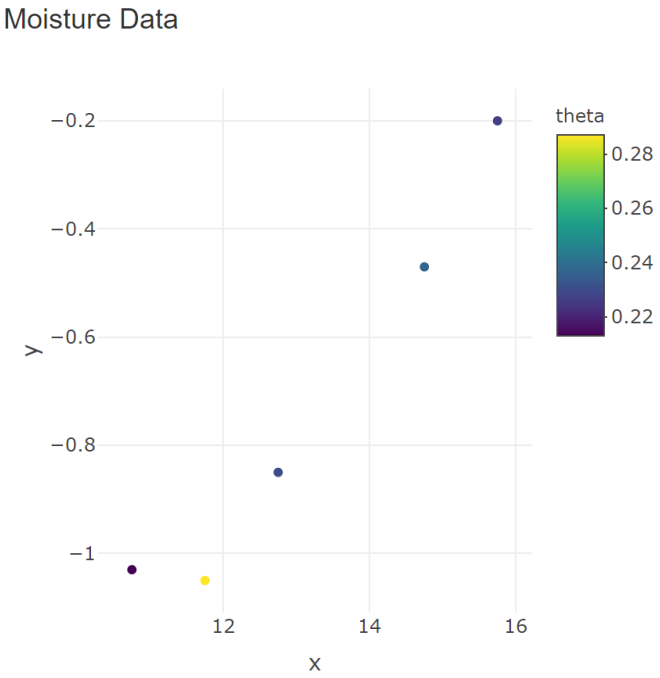
EC data appear in a plot in the lower left once successfully loaded.



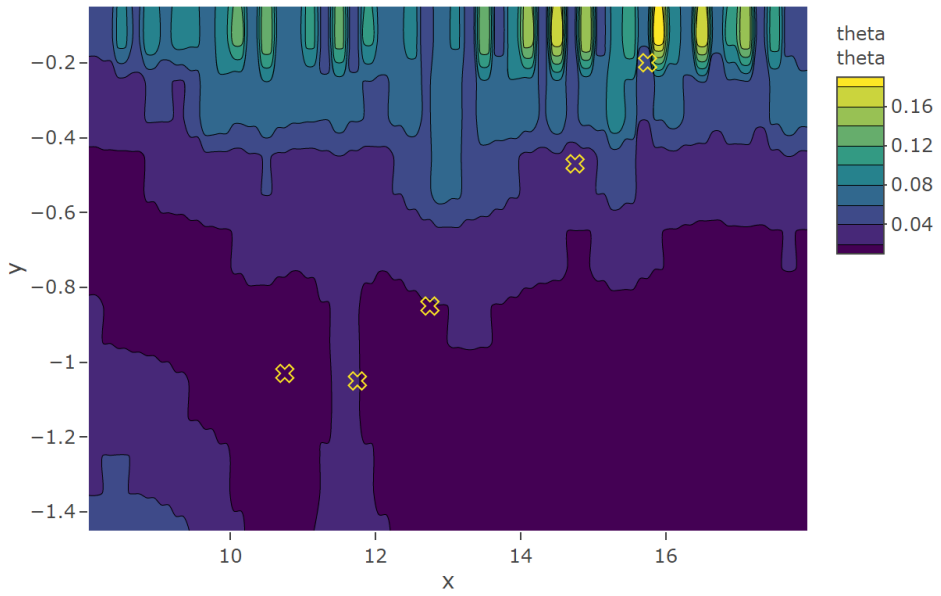
Moisture Data File

A .csv file containing a header with columns X, Y, Z, moisture and % error in moisture

Upon loading this file, point moisture data will be plotted.



When both EC and point moisture data are loaded, a plot at the right shows the combined information from both data types (note this is not the inverse result, but rather just the EC-derived moisture values with point moisture information plotted on top as crosses).



Resolution Data File

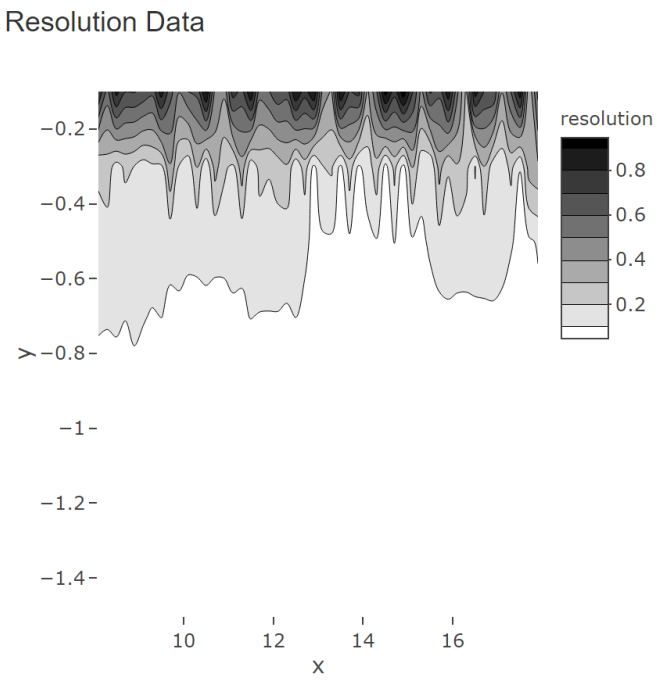
A .csv file containing a header with column X, Y, Z, resolution. Resolution values range from 0 to 1, with 0 indicating no resolving power and 1 indicating maximum resolving power. The resolution information does not need to be of the same size as the data or the model grid – values will be extended to their nearest data point and will be used to scale the data weights.

This is an *ad hoc* approach to adding the effect of variable model resolution from geophysical inversion into the moisture estimation. Some geophysical inversion programs allow for output of a model resolution matrix, whose diagonal elements are considered here. These values, ranging from 0 to 1, are used to scale the data weights in the moisture estimation approach such that weight = resolution/data error. A resolution of 1 therefore has no effect on the data weights, which are influenced only by the data

error. Diminishing resolution, however, also diminishes the weight, until a resolution of 0 effectively gives the data no weight in the estimation.

Input of the resolution matrix is not necessary. The user could, instead of loading a resolution matrix, equivalently adjust the errors in the EC data file prior to the estimation to produce the same result. However, it is important to consider the spatial resolution of the EC data in some way when running this program.

Upon loading the file, the resolution data will be plotted at the bottom of the screen.



Calibration data file and options

A .csv file containing a header with column EC (S/m) and moisture. These data might be derived from a laboratory experiment where one increases moisture of a sample incrementally inside a box where resistivity data may be collected. If a file is selected, then a regression based calibration data will be used to calculate EC moisture values and a plot showing the calibration data will appear in the lower right.

Below the calibration data file are two buttons:

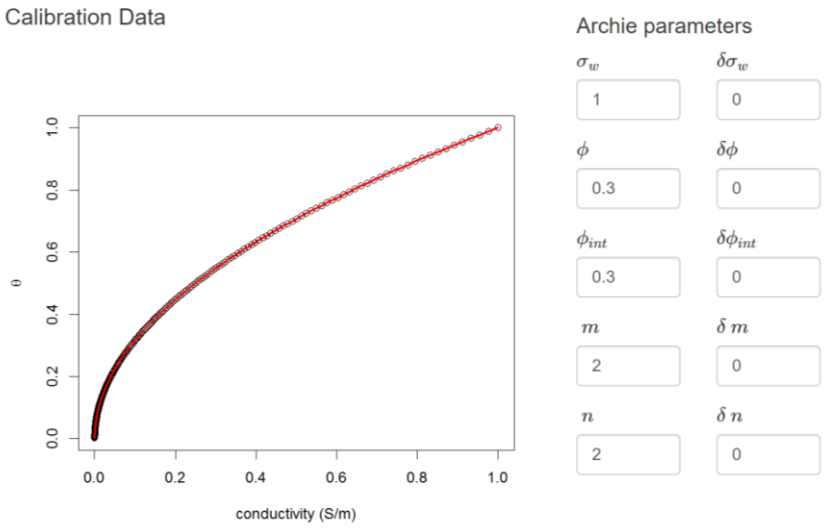
Calibration Data File (optional) ?

Browse... No file selected

use data | use Archie's Law

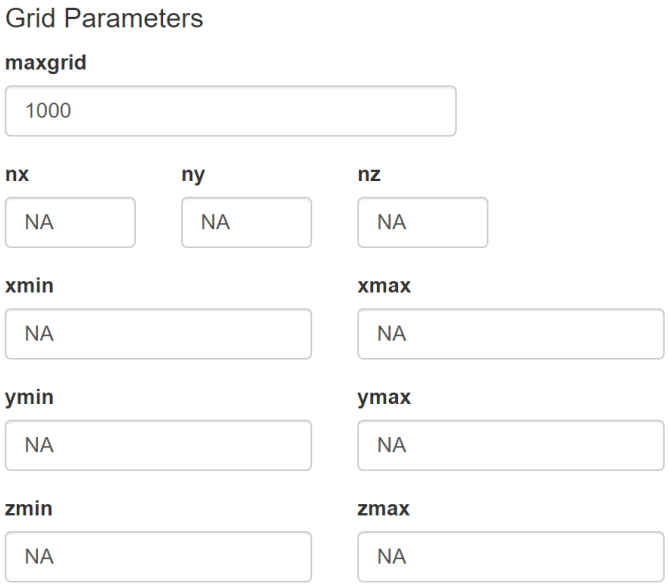
"Use data" uses collocated EC and point moisture information to form a regression. Clicking this button will display a plot in the lower right of EC vs. closest point moisture value, as well as the fit to the data.

The "use Archie's Law" utilizes the Archie parameters at the right of the GUI to convert EC to moisture. The plot in the lower right will display values of moisture computed from Archie's Law for a range of electrical conductivity values. This can be a useful guideline for reasonable Archie values in the case where they are unknown:



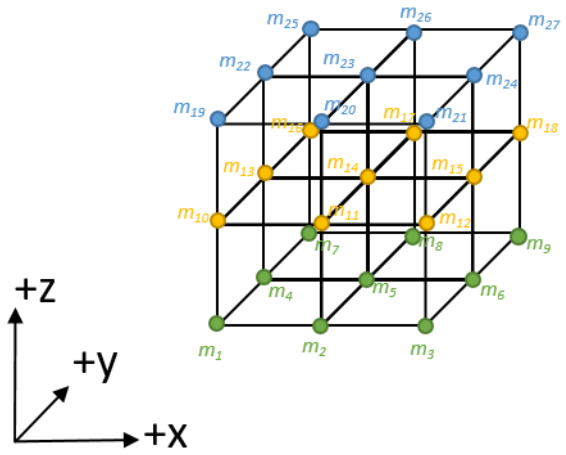
In this example, moisture values greater than the porosity (0.3) are unrealistic. We can see that a conductivity of about 0.1 S/m produces this maximum moisture content. If EC data are higher than 0.1 S/m, the Archie parameters need adjustment.

Grid parameters:



The moisture estimation field is divided into nx x ny x nz grids, evenly spaced between xmin/xmax, ymin/ymax, and zmin/zmax respectively. The user can choose whether to explicitly specify each dimension and bounds, or simply select a maximum number of grid points (maxgrid) to automatically determine equally spaced nodes in each spatial dimension based on data bounds or specified study bounds). Leaving values as NA prompts automatic selection of the option. MoisturEC contains code to handle a mixture of automatic selection and user input for this set of options.

These grid nodes are the model parameters where moisture is to be estimated. They are sorted within the program in increasing order, cycling fastest on x, then y, then z. The figure below shows an example of a model with nx = 3, ny = 3, and nz = 3. Parameters are shown as m_1 through m_{27}.



Calibration options and errors

Archie's Law

This program uses one of three methods to calculate moisture from EC. The first method is using a rearranged version of Archie's Law

$$\theta = \phi \left(\frac{\sigma_b}{(\phi_{int})^m \sigma_w} \right)^{1/n}, \quad (1)$$

θ = moisture content

σ_b = bulk conductivity (EC data)

σ_w = pore fluid conductivity

ϕ = total porosity

ϕ_{int} = interconnected porosity

m = 'cementation' factor which relates to pore geometry

n = exponent describing the variation of resistivity with water saturation

Errors in these parameters can also be included in $\delta\phi$, $\delta\phi_{int}$, $\delta\sigma_w$, δm, δn. These errors, along with the data errors $\delta\sigma_b$, are propagated through Archie's Law using laws of error propagation for multiplication/division and exponents.

Calibration from separate experiment

The second method uses a calibration data file containing θ and σ_b information supplied by the user. The parameters x and n are estimated through a linear regression:

$$\ln(\theta) = \ln(x) + \frac{\ln(\sigma_b)}{n}, \qquad (2)$$

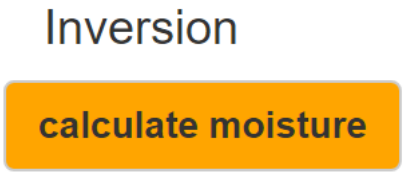
Errors in x and n are determined through actual errors in the regression, and are included as a source of uncertainty in the final moisture estimate.

Calibration from data itself

The final method uses point moisture information and finds the nearest (spatially) EC values to calculate the parameters in Eq. 2 above.

Inversion

Clicking "calculate moisture" will initiate the process of combining the EC-derived moisture content and point moisture.



Moisture content at grid nodes, **m**, is first estimated through a linear solution of the equation,

$$\left[\mathbf{J}^{\mathrm{T}}\mathbf{C}_{\mathrm{D}}^{-1}\mathbf{J} + \alpha\mathbf{D}^{\mathrm{T}}\mathbf{D}\right]\mathbf{m} = \mathbf{J}^{\mathrm{T}}\mathbf{C}_{\mathrm{D}}^{-1}\mathbf{d}, \quad (3)$$

where

d is a column vector of all moisture data. It consists of combined point moisture and EC-derived estimates of moisture content.

J is the Jacobian matrix (M x N). In this case, **J** consists of ones in the rows and columns that correspond to the nearest grid point where data are available; elsewhere zero.

$\mathbf{C}_{\mathrm{D}}$ is a diagonal covariance matrix which consists of the measurement error variances **e** (and $\mathbf{C}_{\mathrm{D}}^{-1}$ are the data weights)

D is regularization matrix (M x M). It consists of a first derivative finite-difference filter between adjacent model nodes.

α is a tradeoff term that controls the balance between regularization criteria and data misfit. Larger values for α promote an increasingly flat solution.

For the first estimate, α is set to 1 (equal trade between smoothing and data misfit). The tradeoff value α is then repeatedly perturbed to find an optimum value for α to use to compute the final set of model parameters, **m**.

"Optimum" means that the misfit between the data itself (EC-derived moisture and point moisture) and the linear solution to Eq. 3 is on the order of errors input to the estimation (which include data errors, errors due to model resolution, and Archie/calibration parameter error).

When complete

Details on estimation of the tradeoff term, α

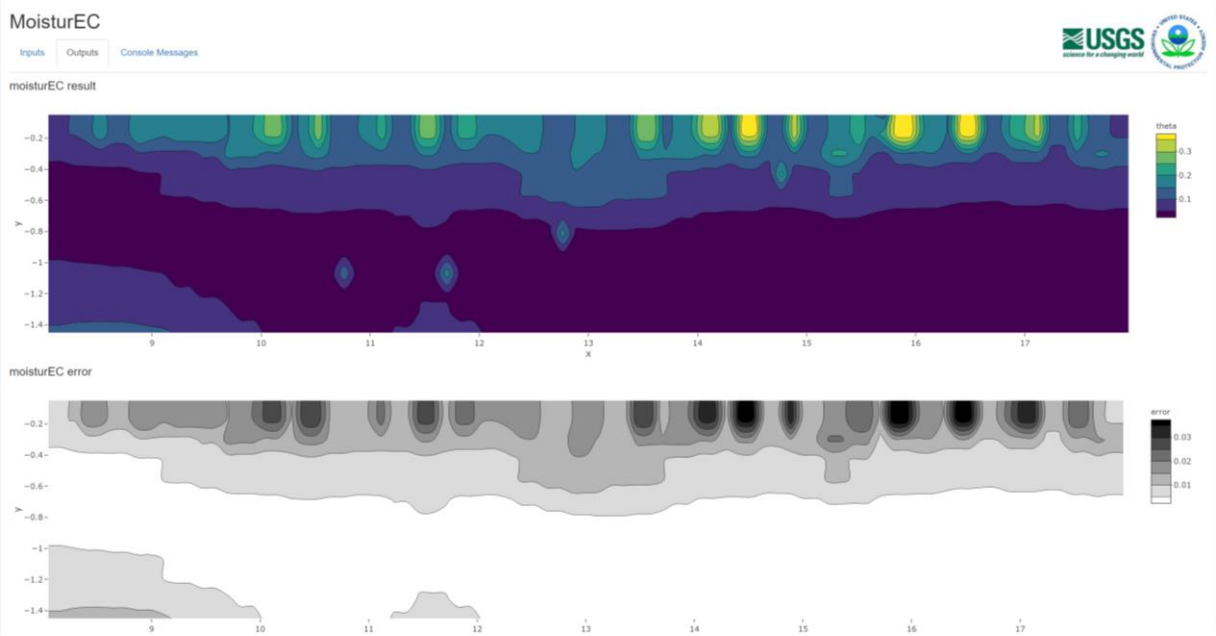
A value for α is achieved through a golden section search and successive parabolic interpolation (**optimize** function in R). The optimize function searches for a minimum value of an objective function, Φ,

$$\Phi = (\chi^2 - 1)^2, \quad (4)$$

which is informed by a chi-squared statistic, χ^2,

$$\chi^2 = \sqrt{\frac{\sum_{i=1}^{n_{data}} \left[\frac{(d_i - \hat{d}_i)^2}{e_i^2} \right]}{n_{data}}}, \quad (5)$$

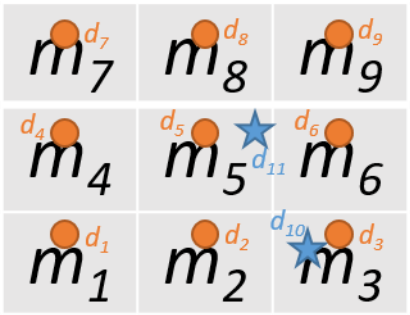
where the $e_i$values are the propagated errors for each EC-derived and point moisture measurement. Thus, a solution is achieved when data misfit values are on the order of the errors, which include the data errors, calibration/Archie errors, and weighting from resolution.



Details of the Jacobian matrix, **J**

The Jacobian is ordinarily a gauge for how much changes in model parameters influence changes in data. In the case of moisturEC, the nearest model cell to each data point is located, and assigned a value of 1 in the Jacobian matrix. All other cells are 0.

Consider the case of a grid of 9 model parameters, with 9 electrical conductivity data points (orange circles) and 2 point moisture data points (blue stars). Note that although model parameters are shown as cells, they are actually grid points in the estimation.



Then the Jacobian matrix looks like this:

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}
m_1	1										
m_2		1									
m_3			1							1	
m_4				1							
m_5					1						1
m_6						1					
m_7							1				
m_8								1			
m_9									1		

Details of the regularization matrix, **D**

Consider the case where we have a 3 x 3 field, for a total of nine model parameters, or nine grid points at which to evaluate moisture content. The model parameters correspond to positions that cycle through horizontal locations first, then vertical locations. The parameters m_1 through m_9 are therefore spatially represented as shown below. Note that although model parameters are shown as cells, they are actually grid points in the estimation.

m_7	m_8	m_9

m_4	m_5	m_6
m_1	m_2	m_3

A 2D spatial finite difference matrix can be constructed as shown below. Here, blank cells correspond to 0 entries and therefore this matrix is handled as a sparse matrix in R using the package Matrix.

Note differencing between adjacent model elements (2 examples highlighted for m_1 and m_8). In moisturEC, construction of this matrix takes place in function 'flatness', which is a custom function written for moisturEC.

"Flatness" first derivative finite difference matrix, **D**, and model parameters, **m**,

2	-1		-1						m_1
-1	2			-1					m_2
	-1	2			-1				m_3
-1			2	-1					m_4
	-1			2	-1				m_5
		-1		-1	2				m_6
			-1			2	-1		m_7
				-1		-1	2		m_8

					-1		-1	2	m_9

A series of finite difference equations results when **D** is multiplied by the vector of model parameters, for example,

$2m_1 - m_2 - m_4$

$2m_8 - m_7 - m_5$

m_7	m_8	m_9
m_4	m_5	m_6
m_1	m_2	m_3

```
flatness = function(nx,ny,nz) {
  #to calculate the first derivative 'flatness' matrix D
  library(Matrix)

  npar = nx*ny*nz

  Dx = Matrix(data = rep(0,npar), nrow = npar, ncol = npar, sparse = T)
  Dy = Matrix(data = rep(0,npar), nrow = npar, ncol = npar, sparse = T)
  Dz = Matrix(data = rep(0,npar), nrow = npar, ncol = npar, sparse = T)


  ex = expand.grid(1:nx,1:ny,1:nz)
  ixind = ex[,1]
  iyind = ex[,2]
  izind = ex[,3]

  inode = 1:(nx*ny*nz)
  ix = ixind[inode]
  iy = iyind[inode]
  iz = izind[inode]

  iwest = inode - 1
  ieast = inode + 1

  inorth = inode + nx
  isouth = inode - nx
```

```
  iup = inode + nx*ny
  idown = inode - nx*ny

  for(loc in 1:npar) {

    if(ix[loc]>=2) {
      Dx[loc,iwest[loc]] = -1
      Dx[loc,loc] = 1
    } else if(nx>1) {
      Dx[loc,ieast[loc]]=-1
      Dx[loc,loc]=1
    }

    if(iy[loc]>=2) {
      Dy[loc,isouth[loc]]=-1
      Dy[loc,loc]=1
    } else if(ny>1) {
      Dy[loc,inorth[loc]]=-1
      Dy[loc,loc]=1
    }

    if(iz[loc]>=2) {
      Dz[loc,idown[loc]]=-1
      Dz[loc,loc]=1
    } else if(nz>1) {
      Dz[loc,iup[loc]]=-1
      Dz[loc,loc]=1
    }

  }

  D=Dx+Dy+Dz
  return(D)
}
```